# Project: ITC Cookie Crush*

Sibt ul Hussain, Amna Irum, Aneeqa Sundus, Atifa Sarwar

**Deadline:** December 8, 2016 before 23h30

## Part I

**Attention**

- Make sure that you read and understand each and every instruction. If you have any questions or comments you are encouraged to discuss your problems with your colleagues (and instructors) on Piazza.
- Plagiarism is strongly forbidden and will be very strongly punished. If we find that you have copied from someone else or someone else has copied from you (with or without your knowledge) both of you will be punished. You will be awarded (straight zero in the project — which can eventually result in your failure) and appropriate action as recommended by the Disciplinary Committee (DC can even award a straight F in the subject) will be taken.
- **Try to understand and do the project yourself even if you are not able to complete the project. Note that you will be mainly awarded on your effort not on the basis whether you have completed the project or not.**
- Divide and conquer: since you have around 10 days so you are recommended to divide the complete task in manageable subtasks. We recommend to complete the drawing and design (*i.e.* number of functions and their calling order) phase as quickly as possible and then focus on the other parts. *[Roughly You will require at least 40 hours to complete the task]*
- **Before writing even one line of code, you must design your final project. This process will require you to break down and outline your program, design your data structure(arrays, stings, etc.), clarify the major functionality of your program, and pseudocode important methods. After designing your program, you will find that writing the program is a much simpler process.**
- *Imagination Powers: Use your imaginative powers to make this as interesting and appealing as you can think of. An excellent solution can get you bonus marks*

**Goals:** In this project you will build a 2D game (ITC's Cookies Crush – see Figure 1 and the provided video and executable (this will work only on Ubuntu 64-bit, you can run this by issuing the command ./cookie-crush in the terminal)) using the techniques learned during the course. The major goal of this project is to consolidate the things you have learned during the course. In this respect it is requested to completely follow the principles you have learned during the course. Moreover, it is an excellent time to put imaginative and creative powers into action.

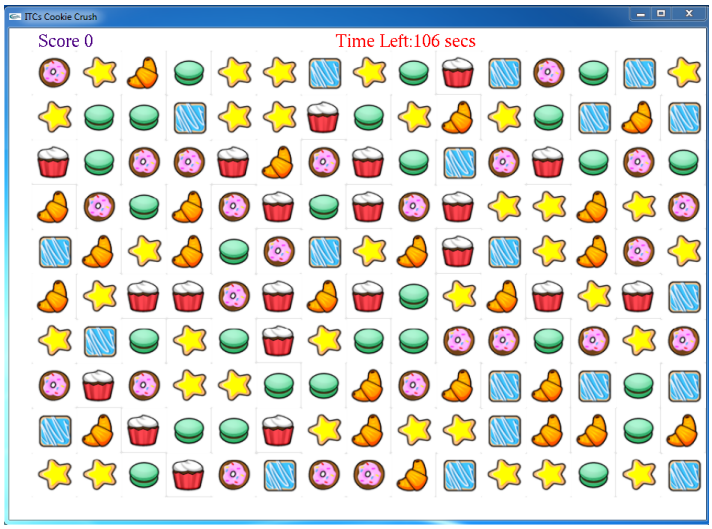


Figure 1: A screen shot of the game.

*Errors and Omissions Expected

# 1 Instructions

We provide complete skeleton with all the basic drawing functions (can be located in util.h and util.cpp) needed in project with detailed instructions and documentation. In other words all you need to know for building the game is provided. Your main task will be to understand the main design of game and implement it. However, before proceeding with code writing you will need to install some required libraries.

## 1.1 Installing libraries on Linux (Ubuntu)

You can install libraries either from the Ubuntu software center or from command line. We recommend command line and provide the file "install-libraries.sh" to automate the complete installation procedure. To install libraries:

1. Simply run the terminal and go to directory which contains the file downloaded file "install-libraries.sh".
2. Run the command

```
bash install-libraries.sh
```

3. Provide the password and wait for the libraries to be installed. If you get an error that libglew1.6-dev cannot be found, try installing an older version, such as libglew1.5-dev by issuing following on command line

```
sudo apt-get install libglew1.5-dev
```

4. If you have any other flavour of Linux. You can follow similar procedure to install "OpenGL" libraries.

## 1.2 Compiling and Executing

To compile the game (skeleton) each time you will be using "g++". However to automate the compilation and linking process we use a program "make". Make takes as an input a file containing the names of files to compile and libraries to link. This file is named as "Makefile" in the game folder and contains the detail of all the libraries that game uses and need to linked.

So each time you need to compile and link your program (game) you will be simply calling the "make" utility in the game directory on the terminal to perform the compilation and linking.

```
make
```

That's it if there are no errors you will have your game executable (on running you will see three shapes on your screen). Otherwise try to remove the pointed syntax errors and repeat the make procedure.

# 2 Drawing Board and Shapes

Your first goal will be to write the code for drawing the canvas and basic shapes.

## 2.1 Canvas

Since we will be building 2D games, our first step towards building any game will be to define a canvas (our 2D world or 2D coordinate space in number of horizontal and vertical pixels) for drawing the game objects (in our case cookies of different shapes). For defining the canvas size you will be using (calling) the function "SetCanvas" (see below) and providing two parameters to set the drawing-world width and height in pixels.

```
/* Function sets canvas size (drawing area) in pixels...
 *  that is what dimensions (x and y) your game will have
 *  Note that the bottom-left coordinate has value (0,0)
 *  and top-right coordinate has value (width-1,height-1).
 *  To draw any object you will need to specify its location
 * */
void SetCanvasSize(int width, int height)
```

## 2.2 Drawing Primitives

Once we have defined the canvas our next goal will be to draw the game board and its elements using basic drawing primitives. For drawing each object we will need to specify its elementary point's locations (x & y coordinates) in 2D canvas space and its size. You will need cookies as drawing primitives to draw the complete board, and its pieces.

For this purpose, skeleton code already include functions for drawing cookies (see below) at specified location.

```
// Drawing functions provided in the skeleton code

/*Draws a specfic cookie at given position coordinate
                  * sx = position of x-axis from left-bottom
                  * sy = position of y-axis from left-bottom
                  * cwidth= width of displayed cookie in pixels
                  * cheight= height of displayed cookiei pixels.
                  * */
void DrawCookie(const Cookies &cname, int sx, int sy, int cwidth = 60,
                int cheight = 60)

// Function draws a string at given x,y coordinates
void DrawString(int x, int y, int width, int height, const string &score,
float*color)
```

Figure 2: A set of functions for drawing primitive shapes.

Skeleton also provides a list of ≈ 140 colors (see file util.h) which can be used for drawing strings of different colors – note that each color is combinations of three individual components red, green and blue and each color is stored as a separate row in the two dimensional array.

## 2.3 Drawing Board

Initially it might seem drawing and managing the board is extremely difficult however this difficulty can be overcome using a very simple trick of divide and conquer. The trick revolve around the idea of the board being split into tiles. "Tile" or "cell" in this context refers to an $30 \times 30$ – you can use any tile size as you wish – pixel square region on the screen. Game's screen resolution is $660 \times 910$ (there are 910 pixel horizontally in each row and there are 660 such rows), so this gives us a total board size of $22 \times 31$ tiles. So drawing and managing the board will require these two steps:

1. Splitting the board in tiles.
2. Finding and storing what part of piece to draw in each tile.

Furthermore you might need to convert pixel-coordinates to cell (or tile)-coordinates and vice versa. It will be extremely helpful for you if you can define two functions for performing these tasks.

Remember that you can do your drawing only in the `Display()` function, that is only those objects will be drawn on the canvas that are mentioned inside the Display function. This `Display` function is automatically called by the graphics library whenever the contents of the canvas (window) will need to be drawn *i.e.* when the window is initially opened, when it is maximized or minimized, and likely when the window is raised above other windows and previously obscured areas are exposed, or when `glutPostRedisplay() function` is explicitly called.

In short, `Display` function is called automatically by the library and all the things inside it are drawn. However whenever you need to redraw the canvas you can explicitly call the Display() function by calling the function `glutPostRedisplay()`. *For instance, you will calling the `Display` function whenever you wanted to animate (move) your objects; where first you will set the new positions of your objects and then call the `glutPostRedisplay()` to redraw the objects at their new positions. Also see the documentation of `Timer` function.*

## 2.4 Interaction with the Game

For the interaction with your game you will be using your mouse. You will need to know when and where the left-mouse button is clicked and released inside in the window of your game. Graphics library will call your corresponding registered functions whenever mouse button is clicked. In the skeleton code we have registered a function (see below) to graphics library. This function is called whenever either mouse button is pressed or released(see the skeleton for complete documentation). Your main tasks here will be to add all the necessary functionality needed in these two functions to to make the game work.

```
/*This function is called (automatically) whenever your mouse button is clicked within inside the game window
 *
 * You will have to add the necessary code here for shooting, etc.
 *
 * This function has four arguments: button (Left, Middle or Right), state (button is pressed or released),
 * x & y that tells the coordinate of current position of move mouse
 *
 * */
void MouseClicked(int button, int state, int x, int y);
```

### 2.5 Game Rules:

- You should fill the board with cookies by choosing their shape randomly under the condition that three cookies of same shape cannot exists among consecutive tiles.
- Cookies can be swapped only to consecutive tiles i.e. top, bottom, right or left.
- Swapping is only allowed if after swapping we have a combination of three or more than three cookies of same shape. In such case the cookies of same shape will get crushed.
- But if on swapping we do not have a combination of three or more than three cookies of same shape than the swapped cookies will automatically be moved back to its original position.
- Whenever the cookies get crushed, the cookies will be dropped from top to the empty slots.
- One point is awarded on crushing of the one cookie.
- The game will be played for only 2 min. After that, the cookies will be disappeared from the screen and user will be shown with game over status.
- When you fill the board randomly, then make sure that if we have a combination of three or more than three cookies of same shape than these cookies should be crushed automatically. Same applies when you drop cookies from the top.

## 3 TimeLine

You have to submit your project in multiple phases. Following is the timeline for the submission of your project:

- Drawing of Board with random shaped cookies as well as auto crushing while filling the board: 30th Nov 2016
- Swapping of cookies, crushing if combination exists, dropping the cookies from the top: 4th December 2016
- Complete Project along with score calculation, auto crushing of the cookies and adding the timer: 8th December 2016

## 4 Marks

You can set your own mile stones however you code will be evaluated keeping in view following guidelines. Remember these are only guidelines, final decision will be of the teacher that how he evaluates your work. Once again remember you will be marked for your understanding, motivation and hard work.

1. Understanding the problem.
2. Identifying the game functionality, its division into manageable functions and their interactions. [**Marks =2 to 3 Absolutes**]
3. Drawing of Board and its pieces. [**Marks = Upto 1-2 Absolute**]
4. Good design and programming practices (such as commenting habits, naming conventions, indentations, *etc.*). [**Marks = Upto 2 Absolutes**]
5. Complete Game with full functionality. [**Marks = Upto 2 Absolutes**]
6. Demonstration quality and answers to the questions. [**Marks = Upto 1 Absolute**]
7. Out of box thinking, imaginative solution, *etc.* [**Bonus Marks = Upto 2 Absolutes, given only if game is working fully**]

Please note that the final authority rests with the instructor and he has all the rights to give you straight zero based on your design or coding habits.

Good Luck :)